UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Modelos evolutivos en cáncer: usando R y C++ para simular intervenciones

Autor: Diego Chicote González
Tutor: Ramón Díaz Uriarte
Ponente: Iván González Martínez

junio 2018

MODELOS EVOLUTIVOS EN CÁNCER: USANDO R Y C++ PARA SIMULAR INTERVENCIONES

Autor: Diego Chicote González
Tutor: Ramón Diaz Uriarte
Ponente: Iván González Martínez

Dpto. de Bioquímica
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio 2018

Diego Chicote González
Modelos evolutivos en cáncer: usando R y C++ para simular intervenciones

Diego Chicote González

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A pesar de las caidas...

Why do we fall?
So that we can learn to pick ourselves up.

Thomas Wayne

Agradecimientos

A mis padres y al resto de mi familia por todo el apoyo que me han dado.

A Laura por su eterna paciencia y constante apoyo.

A Ramón Diaz Uriarte, por su implicación con este TFG.

A Fran y Fer por ser unos compañeros inigualables.

A todos aquellos que me han ayudado en algún momento.

A Eloy Anguiano por su aportación a este TFG con el estilo de este documento.

Resumen

El cáncer se desarrolla por una o varias mutaciones que provocan un crecimiento descontrolado de las células mutadas. Para tratarlo se utilizan diferentes técnicas como la cirugía, la quimioterapia o la radioterapia cuya finalidad es la de eliminar las células cancerosas.

OncoSimulR es un programa que permite simular procesos evolutivos con reproducción asexual, especialmente enfocada a la simulación de procesos tumorales. En el desarrollo de este proyecto se ha trabajado en la extensión de este programa añadiendo una funcionalidad que simula intervenciones quirúrgicas. Esta funcionalidad se ha implementado dentro de OncoSimulR utilizando R, C++.
Las intervenciones permiten que una vez se alcance un tamaño de población determinado en la simulación, el "trigger" de la intervención se dispare y realice la acción estipulada, en este proyecto la acción implementada ha sido la eliminación de un porcentaje de la población, simulando los efectos que produce una cirugía.

Una vez se completó la implementación se realizaron las pruebas necesarias para detectar posibles errores o detalles que no se consideraron previamente. Tras verificar que todo funcionaba correctamente se hicieron simulaciones con dos modelos de crecimiento. Se ejecutaron diferentes casos con y sin intervenciones, variando el porcentaje de eliminación y de esta forma se obtuvieron resultados que permitieron analizar los efectos de las intervenciones en la simulación.

Finalmente se expone de forma general el proceso seguido durante el diseño y desarrollo del proyecto. Se comenta también de cara al futuro las posibles mejoras que se podrían desarrollar en el código.

Palabras clave

cáncer, simulación, intervención, "trigger", acción

Abstract

Cancer begins with one or more mutations that causes uncontrolled growth of mutated cells. To be treated it can be used differents techniques like surgerym chemotherapy, radiation therapy whose purpose is to delete cancerous cells.

OncoSimulR is a forward-time genetic simulator in asexually reproducing populations specially focussed on tumoral processes. In the development of this project work has been done in the extension of this program adding a functionality that simulate surgical interventions. This functionality has been implemented within OncoSimulR using R, C++.
Interventions allow that once a certain population size is reached in the simulation, the trigger of the intervention is triggered and the action stipulated performed, in this case the action to be carried out has been the deletion of a population percentage, action similar to which occurs in a surgery.

Once all the implementation is complete we will test it running tests to detect problems or not considered details. After verify that all works correctly it will be simulated diferent cases with and without interventions with two different growth models.

Finally, the process followed during the project design and development is analyzed in a general way. It will also be comment the possible improvements that could be developed in the project in the near future.

Keywords

cancer, simulation, intervention, trigger, action

Índice

Listas

Lista de figuras

Lista de tablas

1

Introducción

Durante el desarrollo de este capítulo se describirán los detalles básicos del proyecto, es decir, todos aquellos aspectos que han llevado al desarrollo del mismo, los objetivos que se esperan alcanzar una vez finalizado el mismo y se darán indicaciones sobre la organización de los elementos de este documento.

1.1. Motivación del proyecto

El **cáncer** [1] es una enfermedad que se origina cuando tras una o varias mutaciones, las células comienzan a crecer de forma descontrolada. Una vez iniciado este proceso de crecimiento es posible que se forme un tumor. El tumor se puede diseminar por el resto del cuerpo produciendo lo que se conoce como **metástasis**.

Hasta el siglo XX cada persona que padecía esta enfermedad no tenía muchas esperanzas de sanar, sin embargo, al comienzo del siglo XX Paul Ehrlich (1854-1915) [2] recibió el Premio Nobel en Fisiología o Medicina por sus estudios con compuestos químicos para crear medicamentos contra enfermedades, esto le convirtió en el precursor de la quimioterapia. Posteriormente, otros científicos continuaron con la investigación de este campo y continuaron desarrollando estos estudios. A partir de la segunda mitad del siglo XX, la medicina siguió avanzando de forma continuada hasta la actualidad. Hoy en día los médicos ya son capaces de aplicar tratamientos que consiguen aumentar la probabilidad de recuperación del paciente; quimioterapia, radioterapia y cirugía son los ejemplos más comunes, todas ellas basadas en la eliminación completa o parcial de las células cancerosas [3]. En la Figura 1.1 se muestra un ejemplo de como son los cambios en el número de células al realizar una cirugía eliminando gran parte de las células tumorales, después aplicar quimioterapia varias veces y por último realizar de nuevo una cirugía con la que se elimina el 100 % de las células.

No obstante, ningún tratamiento asegura una probabilidad del 100 % de recuperación, pero si podemos desarrollar un sistema de estudio para deducir cual es la opción de entre todos los tratamientos que sería más eficaz o más adecuada para cada caso particular.

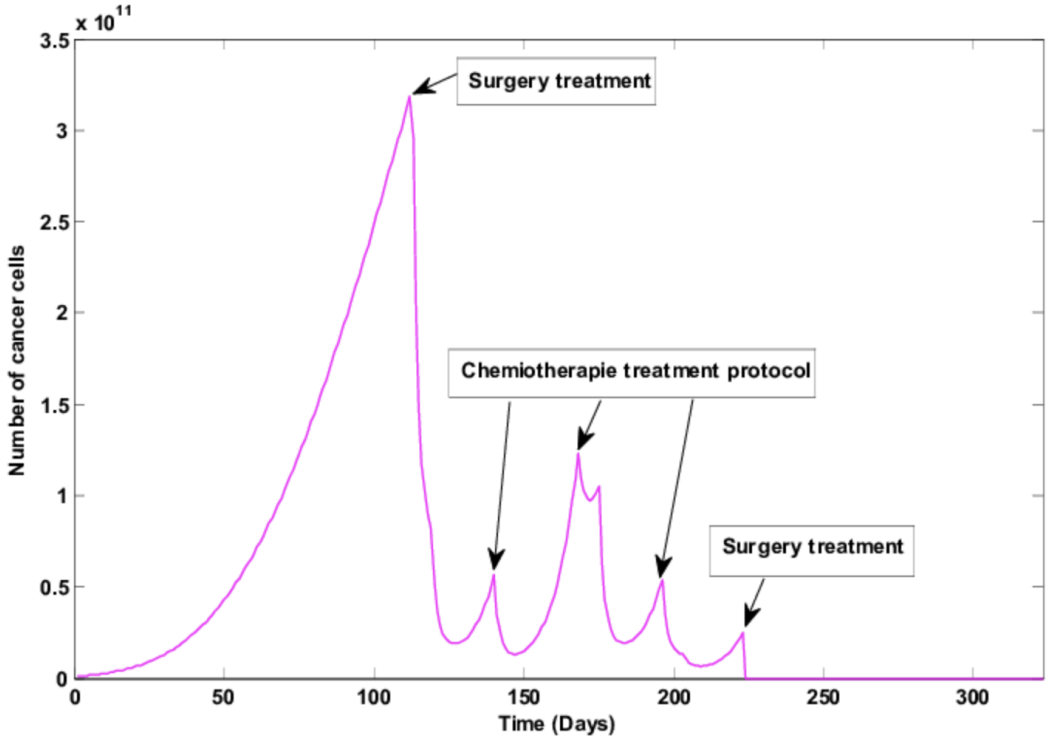


Figura 1.1: Evolución del numero de células tras cirugía y quimioterapia. [4]

La motivación para el desarrollo de este proyecto es poder realizar modificaciones en la población de células durante la simulación de un crecimiento tumoral de tal forma que se pueda observar las consecuencias de estas modificaciones en el desarrollo del tumor. Con esto lo que se quiere conseguir es la simulación de los efectos de los tratamientos contra el cáncer, concretamente este proyecto se centra en la simulación de un efecto equivalente al de una cirugía, permitiendo así tener una aproximación de como evolucionará el tumor tras la operación.

1.2. Objetivos y enfoque

El objetivo principal de este proyecto es la implementación de una funcionalidad que permita simular la aplicación de una o varias intervenciones quirúrgicas durante el desarrollo de la simulación de un tumor de tal forma que se puedan realizar modificaciones en la población de células tumorales en función de una determinada condición.

Los otros objetivos que se han propuesto para el desarrollo del proyecto han sido:

- **Adquirir conocimientos en R, C++:** ya que serán los lenguajes en los que se realizará todo el proyecto.
- **Comprender el funcionamiento de OncoSimulR:** es el programa al que se le añadirá el nuevo código.
- **Añadir las intervenciones a OncoSimulR:** la finalidad principal del proyecto.
- **Comprender los modelos de crecimiento (Exponencial y McFarland):** dado que serán los utilizados durante las simulaciones.
- **Interpretar y utilizar los resultados producidos por la nueva funcionalidad:** una vez implementados los cambios, será necesario interpretar y poder comprobar la corrección de los resultados.
- **Testar la nueva funcionalidad:** es necesario comprobar que todo el código realizado funciona correctamente y no crea problemas con el código base.

Cumplir cada objetivo enumerado es necesario para la correcta finalización del proyecto.

1.3. Organización de la memoria

Este documento consta de los siguientes capítulos que se describirán brevemente a continuación:

- **Estado del arte:** En este capítulo se analizarán diferentes tecnologías similares a la utilizada en este proyecto describiendo su funcionalidad para poder realizar una comparativa cualitativa con respecto a la tecnología elegida.
- **Diseño y Desarrollo:** Se describirá y desarrollará todo lo relacionado con las decisiones de diseño e implementación tomadas. Se explicará en detalle el flujo y el funcionamiento de la nueva funcionalidad implementada. En este capítulo se comenzará explicando todo lo necesario para la realización del proyecto, desde el entorno utlizado hasta los lenguajes de programación usados. En la última parte se detallará todo lo relacionado con la parte de la implementación y uso de las intervenciones.
- **Pruebas y resultados:** Se mostrarán las pruebas realizadas sobre todas las funciones añadidas demostrando su correcto funcionamiento y por otro lado se presentarán los resultados obtenidos mostrando varios casos y realizando diferentes comparativas que muestren la diferencia de funcionamiento tanto con la nueva funcionalidad como sin ella.
- **Conclusiones y trabajo futuro:** Para finalizar se expondrán las conclusiones alcanzadas tras la realización del proyecto. Y se realizaran propuestas de continuación del trabajo de cara al futuro.

2

Estado del arte

Durante el desarrollo de este capítulo se comenta la situación actual de la tecnología describiendo aplicaciones similares a la que se desarrolla en este proyecto. De esta forma se intenta situar en qué punto de desarrollo se encuentra la tecnología y más en concreto en qué situación se encuentra la tecnología que abarca el proyecto.

2.1. TTP: Tool for Tumor Progression

En primer lugar se describirá la aplicación **TTP: Tool for Tumor Progression**. Esta aplicación está implementada en Java y ha sido desarrollada por Johannes G. Reiter, Ivana Bozic, Krishnendu Chatterjee y Martin A. Nowak [5].

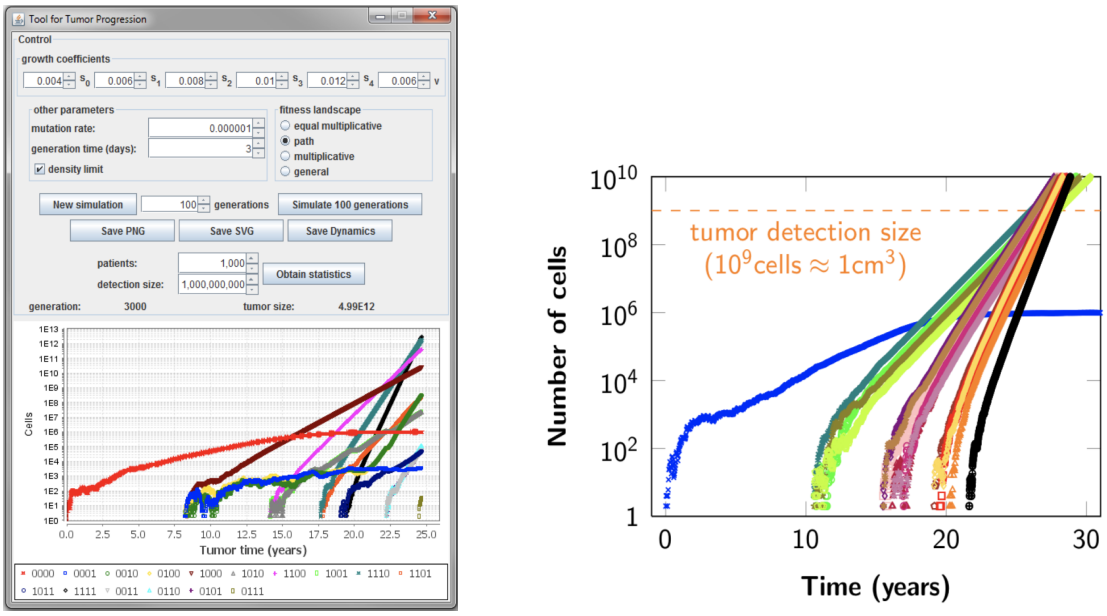
TTP dispone de dos modos de uso:

1.– **Modo individuo:** en este modo simula el crecimiento de un tumor en un paciente durante un número de épocas dado. Permite también la representación de gráficas con los resultados y almacena en un fichero el historial de crecimiento.

2.– **Modo estadístico:** simula un número dado de pacientes, todos ellos con las mismas características y a su vez va mostrando diferentes parámetros relevantes durante la simulación como por ejemplo la media de detección de tumores. También se guardan los resultados en un fichero.

En el primer modo se pueden ajustar 6 parámetros de entrada mientras que en el segundo hasta 8. Como futuros cambios o actualizaciones los desarrolladores quieren poder modelizar los tratamientos contra el cáncer.

Por último en la Figura 2.1 se muestran la interfaz gráfica de la aplicación junto con un ejemplo de ejecución.



(a) Muestra de la interfaz de TTP.

(b) Gráfica resultante de ejecución de TTP.

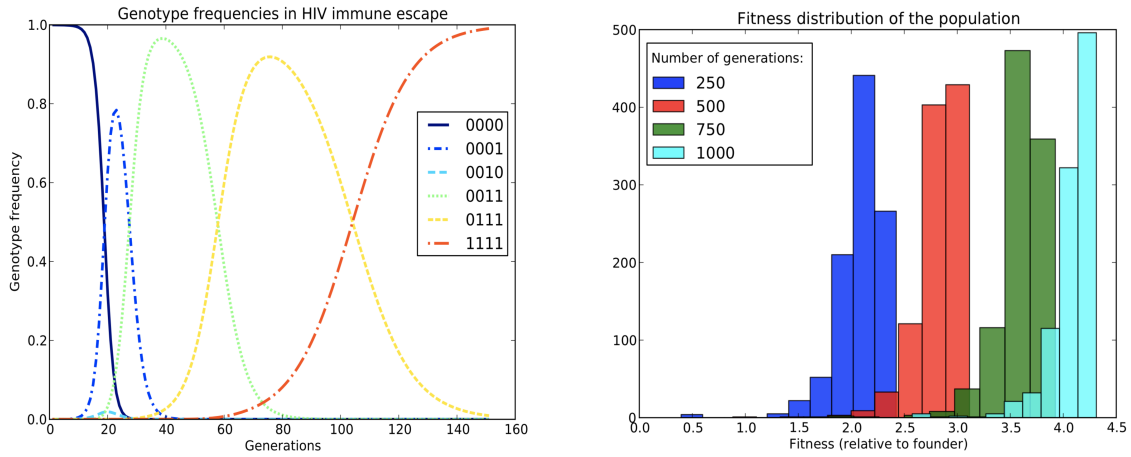
Figura 2.1: Imágenes de TTP.

2.2. FFPopSim: Fast Forward Population Genetic Simulation

FFPopsim: Fast Forward Population Genetic Simulation es una librería para la simulación en genética de poblaciones. Ha sido desarrollada por Fabio Zanini y Richard A. Neher [6]. Está codificada en C++ y utiliza una interfaz en Python. Consta de dos partes, una para realizar simulaciones basadas en un individuo y otra para hacer simulaciones basadas en el genotipo. Las características principales de estas partes son las siguientes:

- **Simulaciones basadas en el genotipo:**
  - Clase haploid_lowd de la librería.
  - Controla las frecuencias de todos los genotipos posibles.
  - Válido para poblaciones asexuales y sexuales.
  - Tiempo pequeño de ejecución.
  - Las simulaciones pueden ser deterministas o estocásticas.
- **Simulaciones basadas en un individuo.**
  - Clase haploid_highd de la librería.
  - Utiliza genomas y poblaciones de gran tamaño.
  - Válido para poblaciones asexuales y sexuales.

En la Figura 2.2 se muestran un par de ejemplos de la salida de FFPopSim.



(a) Frecuencia de los genotipos.

(b) Distribución del fitness.

Figura 2.2: Imágenes de FFPopSim.

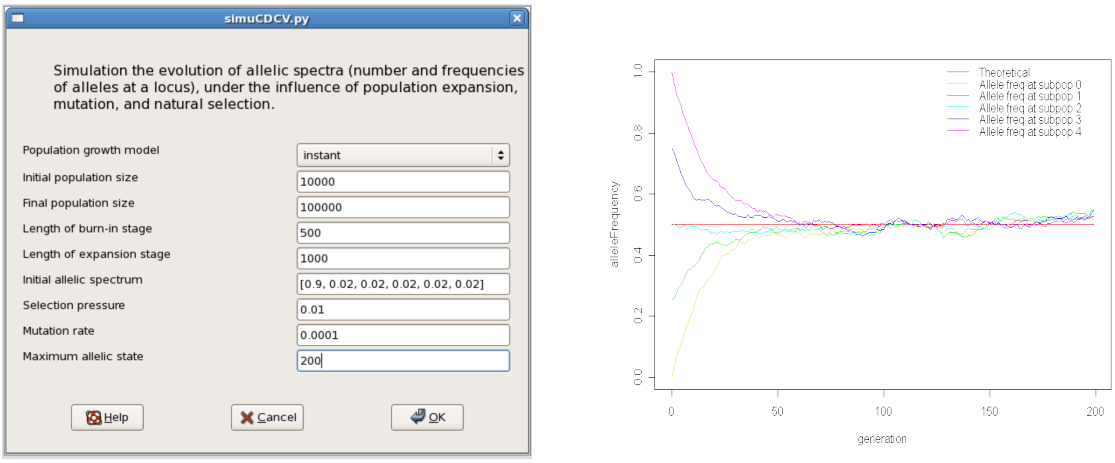
2.3. SimuPOP

La última aplicación que se va a describir es **simuPOP** desarrollada por Bo Peng [7]. Está implementada en C y C++ bajo una interfaz en Python. SimuPOP es un entorno de simulación basado en genética de poblaciones que simula la transmisión de un genotipo cuando la población evoluciona generación a generación. Es de propósito general ya que no está diseñado para un escenario concreto.

Algunas de las características más importantes de simuPOP son:

- Dispone de una gran cantidad de objetos y funciones para el control de la simulación.
- Puede paralelizarse.
- Puede generar y calcular un gran número de estadísticas sobre la simulación.
- Extensa documentación y gran número de ejemplos.

Un ejemplo de la interfaz gráfica y una simulación de simuPOP se muestran en Figura 2.3.



(a) Interfaz gráfica simuPOP.

(b) Simulación de simuPOP.

Figura 2.3: Imágenes de simuPOP.

Estas tres aplicaciones permiten la realización de simulaciones tanto a partir de individuos como a partir de genotipos y todas tienen características muy similares. En los tres casos se dispone de interfaz gráfica facilitando la utilización y todas muestran los resultados mediante gráficas. Cabe destacar que el propósito de simuPOP es más general que el de las otras dos aplicaciones que se han diseñado para fines más específicos.
En el próximo apartado se describe OncoSimulR, la aplicación sobre la que se trabaja en este proyecto. Al final del mismo se expondrán las ventajas de esta aplicación sobre las aplicaciones ya descritas.

2.4. OncoSimulR

Bioconductor es un proyecto de código abierto que se basa en el desarrollo de herramientas para el estudio de datos genómicos. En su mayoría los paquetes de Bioconductor están implementados en R.

OncosimulR es un paquete de Bioconductor con numerosas funcionalidades, entre ellas se destaca la simulación de modelos de crecimiento tumoral, que será la funcionalidad sobre la que se trabajará durante el proyecto. Este paquete ha sido creado y desarrollado por el profesor de la Universidad Autónoma de Madrid, **Ramón Diaz-Uriarte** [8] [9] [10] [11]. Está escrito en R y C++, códigos conectados mediante el paquete de R **Rcpp**. Con respecto a los lenguajes utilizados en este proyecto se hablará más en detalle en el Apartado 3.2.

A continuación se describirán las funciones de OncoSimulR que permiten realizar las simulaciones destacando los argumentos más relevantes y mostrando ejemplos de ejecución de estas funciones. Las funciones destinadas para este propósito son:

- **oncoSimulIndiv():** esta función utiliza el algoritmo BNB [12] para simular la evolución tumoral. Gracias a la numerosa cantidad de argumentos que utiliza esta función permite que el usuario especifique con gran precisión los detalles de la simulación. Algunos de los argumentos más relevantes de esta función son los siguientes:
  - **model:** modelo de crecimiento a utilizar en la simulación puede tomar 3 valores: "Exp": modelo exponencial, "McFL": modelo de McFarland o "Bozic".
  - **mu:** velocidad de mutación.
  - **detectionSize:** número de células necesarias para detectar cáncer.
  - **sampleEvery:** se indica cada cuanto se muestrea la población.
  - **keepEvery:** intervalo del tiempo entre los que se guarda el muestreo de la población.
  - **initSize:** tamaño inicial de la población.
  - **finalTime:** unidades de tiempo máximas a ejecutar.
  - **onlyCancer:** indica si se desea sólo simulaciones que detectan cáncer.

  Un ejemplo de llamada a oncoSimulIndiv() sería el que se muestra en **Código 2.1**:

Código 2.1: Ejemplo de utilización de la función oncoSimulIndiv()

```
1  nd <- 70
2  np <- 0
3  s <- 0.1
4  sp <- 1e-3
5  spp <- -sp/(1 + sp)
6  exp <- allFitnessEffects(noIntGenes = c(rep(s, nd), rep(spp, np)),
7                           drvNames = seq.int(nd))
8  exp <- oncoSimulIndiv(mcf1,
9                        model = "Exp", #McFL | Exp
10                       mu = 1e-5,
11                       detectionProb = NA,
12                       detectionSize = 1e4,
13                       detectionDrivers = NA,
14                       sampleEvery = 0.1,
15                       keepEvery = 0.1,
16                       initSize = 1000,
17                       finalTime = 2000,
18                       onlyCancer = FALSE)
```

Tras la ejecución de oncoSimulIndiv() esta nos devuelve una lista con elementos de tipo "oncosimul" que contiene diferentes elementos como los que se muestran en la Figura 2.4.

```
  NumClones TotalPopSize LargestClone MaxNumDrivers MaxDriversLast NumDriversLargestPop TotalPresentDrivers FinalTime NumIter HittedWallTime
1       133        10007         4264             3              3                   2                  57      82.5     995          FALSE
  HittedMaxTries errorMF minDMratio minBMratio
1          FALSE      NA   642.1494   833.3333
```

Figura 2.4: Resultado devuelto por la ejecución de oncoSimulIndiv()

Entre los datos que se muestran en la Figura 2.4 cabe destacar alguno como NumClones, número de clones, TotalPopSize, población al final de la simulación, FinalTime, unidad de tiempo al terminar la ejecución, NumIter número de iteraciones del algoritmo BNB.

- **oncoSimulPop():** Llama múltiples veces a oncoSimulIndiv(), Nindiv se le pasa como argumento y es el número de simulaciones a ejecutar. Las simulaciones se pueden paralelizar ya que se le puede pasar por argumento mc.cores que es el número de procesadores a utilizar durante la ejecución. Esto es así gracias a la función mclapply() que paraleliza la ejecución de una función sobre los elementos de una lista, devolviendo la lista con lo resultados.

El código que se ha utilizado para oncoSimulPop() se muestra en **Código 2.2**:

Código 2.2: Ejemplo de utilización de la función oncoSimulPop()

```
1  nd <- 70
2  np <- 0
3  s <- 0.1
4  sp <- 1e-3
5  spp <- -sp/(1 + sp)
6  exp <- allFitnessEffects(noIntGenes = c(rep(s, nd), rep(spp, np)),
7                           drvNames = seq.int(nd))
8  exp2 <- oncoSimulPop(5, mcf1,
9                       model = "Exp", #McFL | Exp
10                      mu = 1e-5,
11                      detectionProb = NA,
12                      detectionSize = 1e4,
13                      detectionDrivers = NA,
14                      sampleEvery = 0.1,
15                      keepEvery = 0.1,
16                      initSize = 1000,
17                      finalTime = 2000,
18                      mc.cores = 1,
19                      onlyCancer = FALSE)
```

En la Figura 2.5 se encuentra el resultado obtenido por esta ejecución.

Los datos mostrados son iguales que en oncoSimulIndiv() sólo que es este caso se muestran más de una fila que se corresponden con el número de simulaciones ejecutadas.

```
  NumClones TotalPopSize LargestClone MaxNumDrivers MaxDriversLast NumDriversLargestPop TotalPresentDrivers FinalTime NumIter HittedWallTime
1       140        10035        4600             3              3                   2                  53      90.7    1092          FALSE
2       137        10018        7484             3              3                   1                  57      79.8     989          FALSE
3       115        10015        7870             3              3                   1                  51      59.5     750          FALSE
4       158        10113        2400             3              3                   1                  60      77.7    1002          FALSE
5       167        10116        3966             3              3                   1                  60      73.4     987          FALSE
  HittedMaxTries errorMF minDMratio minBMratio
1          FALSE      NA   642.1494   833.3333
2          FALSE      NA   642.1494   833.3333
3          FALSE      NA   642.1494   833.3333
4          FALSE      NA   642.1494   833.3333
5          FALSE      NA   642.1494   833.3333
```

Figura 2.5: Resultado devuelto por la ejecución de oncoSimulPop()

Una vez visto lo más representativo de OncoSimulR se pueden especificar las ventajas con respecto de otras aplicaciones similares. Las principales características que hacen que OncoSimulR destaque con respecto de otras herramientas de simulación de cara a la simulación de la evolución tumoral son:

- Gran flexibilidad para especificar el "fitness" y los efectos de las mutaciones.
- Opciones para muestrear y parar las simulaciones. Característica especialmente conveniente en la simulación de modelos evolutivos en cáncer junto con otras posibles opciones que se pueden especificar como los efectos de orden y las restricciones en el orden de acumulación de las mutaciones.
- La posibilidad de simular "fitness landscapes" y evaluar la predictibilidad evolutiva.

Estas características no están disponibles en ninguna otra aplicación de simulación y por lo tanto hacen de OncoSimulR la aplicación idónea para la realización de simulaciones en modelos evolutivos en cáncer.

Tras el análisis de estas cuatro aplicaciones (TTP, FFPopSim, SimuPOP y OncoSimulR) se puede concluir que todas ellas se utilizan para realizar simulaciones de la evolución de distintos tipos de poblaciones de células. Permitiendo a su vez obtener estadísticas y representar el desarrollo de la población mediante el uso de gráficas que posteriormente podrán ser analizas. Ninguna de ellas dispone de una forma de simular los efectos de los tratamientos sobre el cáncer. OncoSimulR es la aplicación con la mayor flexibilidad y versatilidad gracias a la gran cantidad de parámetros que se pueden especificar en la ejecución de sus simulaciones y las numerosas funcionalidades de las que dispone.

3

Diseño y desarrollo

3.1. RStudio Server y Docker

Para el desarrollo del código se ha utilizado RStudio Server sobre un contenedor de Docker. Para ello se ha utilizado la imagen rocker/rstudio [13] que permite con un solo comando disponer de un sistema operativo Debian con RStudio Server ya instalado y listo para usar.

La utilización de RStudio Server da acceso a través de un navegador web a la interfaz de RStudio desde la que se realiza la implementación del código. A parte otras de las ventajas de la utilización de RStudio es que integra ayuda, dispone de auto completado de código y permite la visualización de gráficos directamente en la propia interfaz.

Docker [14] permite crear contenedores portables y ligeros de tal forma que que se pueden ejecutar independientemente del sistema operativo siendo sólo necesaria la instalación de la aplicación de Docker.

Se ha decidido utilizar Docker porque simplifica el proceso de instalación y uso de un sistema operativo paralelo al del host, al poder instalarlo en un contenedor a través de una imagen del sistema operativo elegido. Reduciendo también la cantidad de recursos (memoria, espacio en disco, etc.) a utilizar en comparación con una máquina virtual o la instalación en disco de un sistema operativo. En la Figura 3.1 se muestra la diferencia de arquitectura entre un contenedor de Docker y una máquina virtual.

Por otro lado Docker también da la posibilidad de tener persistencia ya que al cerrar un contenedor se guarda su estado actual y cuando se vuelve a lanzar se encuentra en el mismo estado en el que se encontraba en el momento del cierre. También aporta “repetibilidad” dado que se pueden reproducir copias de un contenedor a partir de un Dockerfile, evitando de esta forma problemas de compatibilidades de cualquier tipo. Esto es importante ya que en algunos casos es necesario que las condiciones en las se realiza una simulación sean iguales en posteriores simulaciones.

Para poder apreciar la simplicidad que aporta Docker se desarrolla en más detalle el proceso de instalación de los componentes necesarios para el desarrollo del proyecto en el Apéndice BAnexo B.

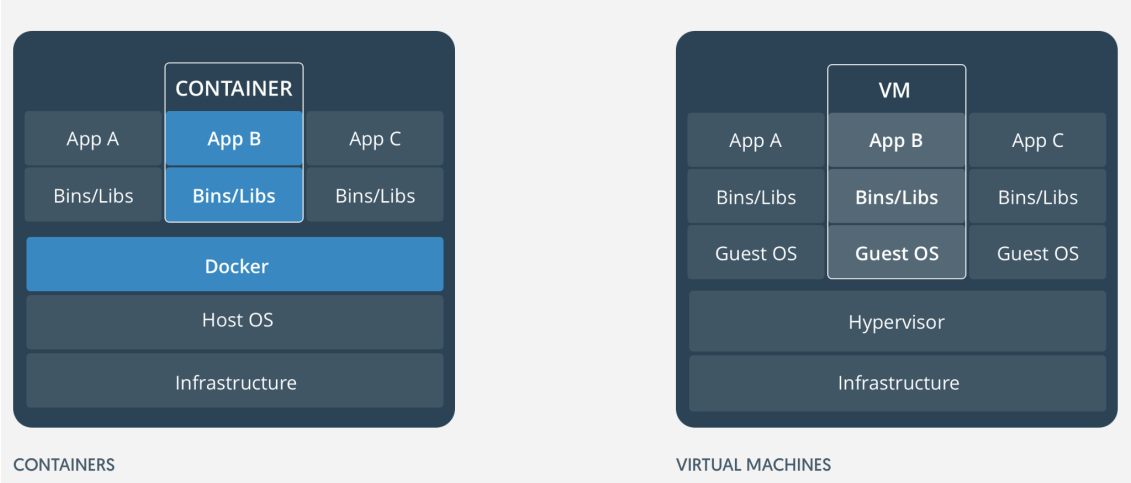


Figura 3.1: Comparación arquitectura contenedor vs maquina virtual.

Hay que destacar que Docker es mucho más potente y tiene un uso mucho más amplio del que se le ha dado en este TFG pero eso queda fuera de los objetivos de esta sección.

3.2. R, C++ y Rcpp

Los lenguajes de programación que se utilizan en este proyecto son R y C++, ya que el paquete OncoSimulR, sobre el que se va a trabajar está implementado utilizando ambos.

Las características que justifican el uso de R como lenguaje principal se enumeran a continuación:

1.– **Idóneo para análisis estadístico:** gracias a su simplicidad y flexibilidad para el desarrollo de este tipo de tareas.

2.– **Potente librería gráfica:** R dispone de numerosos paquetes para este propósito como: ggplot2, googleVis o ggvis.

3.– **Open-source:** permitiendo que cada día aumenten los paquetes que se añaden a CRAN, Bioconductor, etc.

4.– **Multiplataforma:** se puede ejecutar en cualquier sistema operativo.

5.– **Paralelismo:** R también cuenta con funciones y paquetes que permiten paralelizar las ejecuciones. Funciones como mclapply() y paquetes como parallel son ejemplos de ello.

Por otro lado R no es tan bueno para procesos que requieren mucha potencia de computación o necesitan un uso eficiente de los recursos del ordenador, por tanto se utiliza C++ para el código de las simulaciones, ya que es mucho más eficiente y rápido que R. Reduciendo en gran medida los tiempos de ejecución.

Para poder utilizar estos dos lenguajes de forma combinada R dispone de el paquete **Rcpp**. Este paquete permite realizar de forma sencilla llamadas a código C++ desde R y viceversa. También permite que se intercambien objetos de R con C++. Las funciones pueden aceptar y devolver objetos nativos de R lo que hace que el intercambio de datos hacia R sea más sencillo.

A continuación se detalla el cómo se realiza la llamada entre los códigos. Utilizaremos la función *nr_BNB_Algo5()* como ejemplo. Esta función se encuentra implementada en C++ y se declara de la siguiente forma:

Código 3.1: Declaración de la función en C++.

```
1  // [[Rcpp::export]]
2  Rcpp::List nr_BNB_Algo5(Rcpp::List rFE,
3                  Rcpp::NumericVector mu_,
4                  double death,
5                  double initSize,
6                  double sampleEvery,
7                  double detectionSize,
8                  double finalTime,
9                  ... // Dado que son demasiados argumentos para mostarlos se ponen ... para facilitar el
                       visionado
10                 double cPDetect_i,
11                 double checkSizePEvery,
12                 bool AND_DrvProbExit,
13                 Rcpp::List fixation_i,
14                 Rcpp::List modelChanges) { ... }
```

Para que R pueda llamar a esta función se requiere del uso de la función *.Call()* que se encarga de llamar al código de C++ que utiliza los objetos nativos de R. Los argumentos de la función de C++ se pasan como una secuencia de objetos de R.

Código 3.2: Función nr_BNB_Algo5() en el fichero RcppExports.R.

```
nr_BNB_Algo5 <- function(rFE, mu_, death, initSize, sampleEvery,
    detectionSize, finalTime, initSp, initIt, seed, verbosity,
    speciesFS, ratioForce, typeFitness_, maxram, mutationPropGrowth,
    initMutant_, maxWallTime, keepEvery, K, detectionDrivers, onlyCancer,
    errorHitWallTime, maxNumTries, errorHitMaxTries, minDetectDrvCloneSz,
    extraTime, keepPhylog, MMUEF, full2mutator_, n2, p2, PDBaseline, cPDetect_i,
    checkSizePEvery, AND_DrvProbExit, fixation_list, modelChanges) {
    .Call('OncoSimulR_nr_BNB_Algo5', PACKAGE = 'OncoSimulR', rFE, mu_, death,
        initSize, sampleEvery, detectionSize, finalTime, initSp, initIt, seed,
        verbosity, speciesFS, ratioForce, typeFitness_, maxram, mutationPropGrowth,
        initMutant_, maxWallTime, keepEvery, K, detectionDrivers, onlyCancer,
        errorHitWallTime, maxNumTries, errorHitMaxTries, minDetectDrvCloneSz,
        extraTime, keepPhylog, MMUEF, full2mutator_, n2, p2, PDBaseline, cPDetect_i,
        checkSizePEvery, AND_DrvProbExit, fixation_list, modelChanges)
}
```

Una vez hecho esto ya es posible llamar a la función desde R:

Código 3.3: Llamada a la función nr_BNB_Algo5() desde R.

```
nr_BNB_Algo5(rFE = rFE,
             mu_ = mu,
             death = death,
             initSize = initSize,
             sampleEvery = sampleEvery,
             detectionSize = detectionSize,
             finalTime = finalTime,
             ... # Dado que son demasiados argumentos para mostarlos se ponen ... para facilitar el
                 visionado
             cPDetect_i= dpr["cPDetect"],
             checkSizePEvery = dpr["checkSizePEvery"],
             AND_DrvProbExit = AND_DrvProbExit,
             fixation_list = fixation_list,
             modelChanges = modelChanges),
             Drivers = list(rFE$drv),
             geneNames = list(names(getNamesID(rFE))),
             InitMutant = initMutantString
    ))
```

Cuando se ejecute el código C++ este devolverá el resultado en forma de objeto de R.

3.3. Intervenciones

Las intervenciones son el procedimiento por el que simularemos cambios en la población durante el desarrollo de la simulación cuando se cumplan unas condiciones predefinidas.

De forma general una intervención debe estar formada por dos elementos: uno o varios ***"triggers"*** que indicarán el evento que provoca la activación de la intervención (Ej: un tamaño determinado en la población de células), y una o varias ***acciones*** que serán los cambios que lleva a cabo la intervención (Ej: porcentaje de la población a eliminar).

A continuación se especificarán las implementaciones de las intervenciones en los lenguajes utilizados. Las intervenciones se implementan en R como listas de listas de listas. Están formadas por una lista que contendrá todas las intervenciones, que a su vez serán listas compuestas por dos listas, una con los "triggers" y otra con las acciones de cada intervención. Los "triggers" y las acciones se han especificado como listas ya que como se explica en la Apartado 5.2 se ampliaran los tipos de ambos en el futuro y se añadirán como nuevos elementos de la lista. En el siguiente código (**Código 3.4**) podemos ver como se declaran tres intervenciones en R.

Código 3.4: Declaración de intervenciones en R.

```
1 modelChanges = list( Change1 = list( trigger = list(popSize = 3000),
2                                      action = list(fractionPopSize = 0.15)),
3                      Change2 = list( trigger = list(popSize = 5000),
4                                      action = list(fractionPopSize = 0.1)),
5                      Change3 = list( trigger = list(popSize = 7000),
6                                      action = list(fractionPopSize = 0.01)))
```

En C++ se utiliza un vector para almacenar las intervenciones. En este caso las intervenciones son estructuras de tipo ***Intervention*** creadas específicamente para el almacenamiento de las intervenciones procedentes del código en R. Estas estructuras contienen tres elementos: una estructura de tipo ***"Trigger"***, una estructura de tipo ***Action*** y un número como índice de la intervención. Las estructuras "Trigger" y Action actualmente solo disponen de un elemento que se corresponde con el tamaño de la población y el porcentaje de la población a eliminar respectivamente. En caso de añadir más tipos de triggers y acciones se añadirían los atributos necesarios a cada estructura. El (**Código 3.5**) muestra la implementación de las estructuras comentadas en este párrafo.

Código 3.5: Estructuras creadas para las intervenciones en C++.

```
1  struct Trigger { //Trigger structure, in the future add more atributes
2    double popSize;
3  };
4  struct Action {  //Action structure, in the future add more atributes
5    double fractionPopSize;
6  };
7  struct Intervention { //Intervention structure
8    Trigger trigger;
9    Action action;
10   double indx;
11 };
```

Antes de continuar es importante destacar la razón por la que se ha decidido realizar de esta forma la implementación de las intervenciones. La razón principal es que así el código es fácilmente extensible. Permitiendo la ampliación de los tipos de "triggers" y acciones que soportan las intervenciones de manera sencilla. Tomando como ejemplo el **Código 3.4**, si se diera el caso de querer añadir un nuevo "trigger" que se disparara en función del tiempo de ejecución simplemente se tendría que añadir el nuevo campo a la estructura "Trigger" *double time;* y añadirlo a la intervención deseada en este caso a "Change1" *Change1 = list(trigger = list(popSize = 3000, time = 500)...)*. Si se quisiera añadir una acción nueva el proceso sería el mismo pero en la estructura "Action" y en la lista de acciones de la intervención. Se puede observar la flexibilidad de esta implementación y lo fácil que es extender las intervenciones con nuevos tipos.

Una vez explicada en detalle la implementación de las intervenciones en ambos lenguajes se describe el flujo de ejecución que lleva las intervenciones desde su declaración en R hasta las estructura de C++. En este punto es donde entra en juego ***Rcpp*** como se ha explicado en Apartado 3.2. Para que las intervenciones se pudieran pasar a C++ ha sido necesaria la modificación de todas las funciones implicadas en el flujo de ejecución de una simulación, siendo necesario añadir como argumento en todas ellas las intervenciones. Las funciones afectadas han sido *oncoSimulIndiv, oncoSimulPop, nr_ oncoSimul.internal, nr_ BNB_ Algo5, y nr_ totPopSize_ and_ fill_ out_ crude_ P*

Una vez se tienen las intervenciones en C++ es necesario guardarlas en el vector de intervenciones utilizando las estructuras mencionadas anteriormente. Esto se realiza en el **Código 3.6**.

Código 3.6: Código que permite almacenar las intervenciones.

```
1  int i;
2  inters.clear();
3  interv_time.clear();
4  interv_pop.clear();
5  for(i = 0 ; i< modelChanges.size() ; i++){
6    inters.push_back(Intervention());
7    inters[i].trigger = Trigger();
8    inters[i].trigger.popSize = Rcpp::as<double>( Rcpp::as<Rcpp::List>(
9      Rcpp::as<Rcpp::List>(
10       modelChanges[i])["trigger"])["popSize"] );
11
12   inters[i].action = Action();
13   inters[i].action.fractionPopSize = Rcpp::as<double>( Rcpp::as<Rcpp::List>(
14     Rcpp::as<Rcpp::List>(
15       modelChanges[i])["action"])["fractionPopSize"] );
16
17   inters[i].indx = i;
18 }
```

Lo primero que se hace es vaciar el vector de intervenciones para evitar problemas y asegurar que siempre está vacío al inicio. Después se realiza un número de veces igual al tamaño de la lista de intervenciones un bucle for que se encarga de:

1.– Añadir al vector una intervención a través de la estructura ***Intervention***

2.– A la intervención se le añaden el "trigger" y la acción utilizando ***Rcpp::as<double>(Rcpp::as<Rcpp::List>(Rcpp::as<Rcpp::List>...*** que permite pasar un objeto de R a variable de C++ y de esta forma poder guardarlo en la estructura correspondiente.

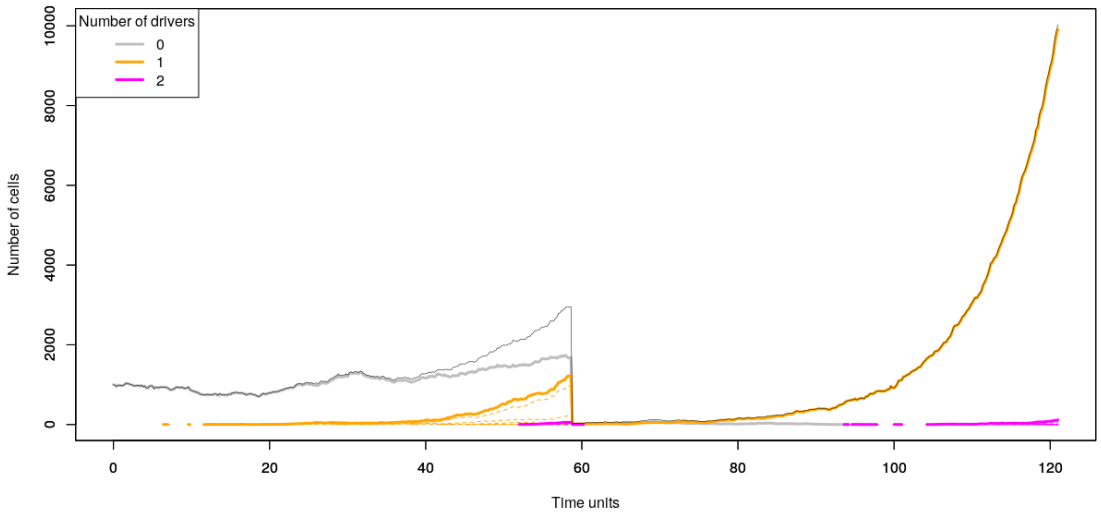
3.– Se asigna el índice de la intervención.

Estos pasos se repiten para todas las intervenciones de la lista. Una vez guardadas en el vector las intervenciones se usarán en el momento necesario. Para ello se recorrerá el vector y se comprobará si alguna de las condiciones de los "triggers" se cumple, en caso de que se cumpla se procede a realizar la acción correspondiente a esa intervención reduciendo la población el porcentaje indicado por la acción modificando los tamaños almacenados de las poblaciones de los clones. Esto se lleva a cabo utilizando un muestreo multinomial [15] ya que se busca muestrear en función de la proporción actual. Puede que la proporción resultante de poblaciones varíe tras la intervención debido al muestreo. Una vez hecho el muestro se asignan los nuevos valores a las poblaciones y se elimina la intervención ejecutada de la lista de intervenciones. Este proceso se repite cada vez que se activa un "trigger" de una intervención. Todos los pasos descritos los lleva a cabo el **Código 3.7**.

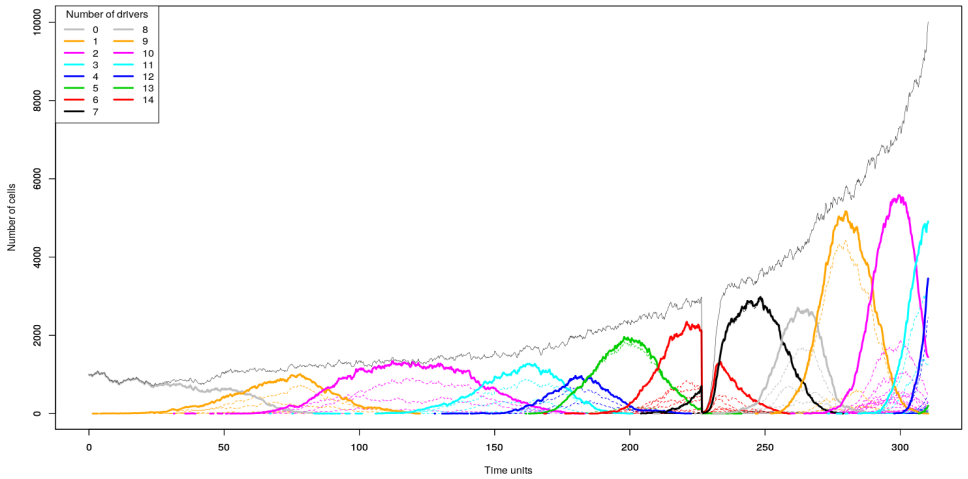
Código 3.7: Código que controla y ejecuta las intervenciones.

```
float tot_pop = 0.0;
  for(size_t i = 0 ; i < inters.size() ; ++i){

    if (totPopSize >= inters[i].trigger.popSize){
      storeThis = true;
      Rcpp::Rcout << "\n" << "Intervention [" << inters[i].indx <<
        "] applied in time " << currentTime << "\n";

      interv_time.push_back(currentTime);
      interv_pop.push_back(totPopSize);

      tot_pop = totPopSize *inters[i].action.fractionPopSize;
      Rcpp::Rcout << "Before: " << totPopSize << " After: " << tot_pop;
      Rcpp::NumericVector probSizes(popParams.size());
      for(size_t j = 0 ; j < popParams.size() ; ++j){
        probSizes[j] = popParams[j].popSize/totPopSize;
      }

      Rcpp::IntegerVector new_PopSizes(popParams.size());

      std::vector<int> sp_to_remove;
      sp_to_remove.clear();
      new_PopSizes = rmultinom((int)tot_pop, probSizes);

      totPopSize = 0.0;
      for(size_t k = 0 ; k < popParams.size() ; ++k){
        if ((double) new_PopSizes[k] == 0.0){
          sp_to_remove.push_back(k);
        }
        popParams[k].popSize = (double) new_PopSizes[k];
        totPopSize = totPopSize + (double) new_PopSizes[k];
      }
      Rcpp::Rcout << "\n After multinomial: " << totPopSize;
      if(sp_to_remove.size())
        remove_zero_sp_nr(sp_to_remove, Genotypes, popParams, mapTimes);

      inters.erase(inters.begin() + i);
      break;
    }

  }
```

Al ejecutar una simulación con una intervención los resultados obtenidos para los modelos Exponencial y de McFarland serían los que aparecen en la Figura 3.2



(a) Resultado de la ejecución de la simulación con el modelo Exponencial realizando una intervención.



(b) Resultado de la ejecución de la simulación con el modelo de McFarland realizando una intervención.

Figura 3.2: Gráficas obtenidas.

En las gráficas de la Figura 3.2 las líneas de colores representan los clones con n genes mutados (0, 1, 2 en el caso de (a)) mostrando su evolución para dos modelos de crecimiento distintos (Exponencial y McFarland) aplicando una intervención al llegar a tamaño de población 3000. En las gráficas se ve como la intervención provoca el descenso abrupto del tamaño total de la población (linea negra más fina) y posteriormente se aprecia como van recuperándose las poblaciones en ambos casos y continúan creciendo.

En cuanto al modelo de McFarland se presentó la siguiente situación: Tras ejecutar una simulación con las intervenciones por primera vez se obtiene la gráfica de la Figura 3.3.

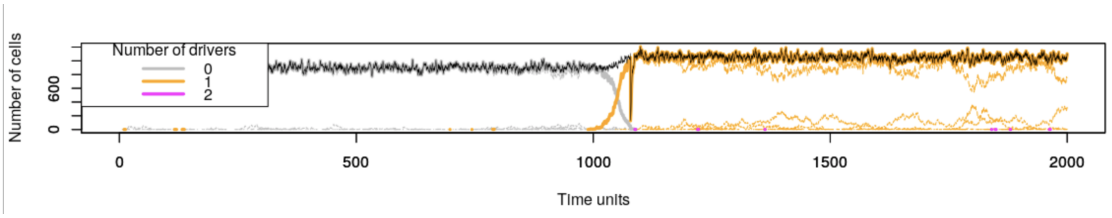


Figura 3.3: Resultado de la primera simulación con intervenciones.

A la vista del resultado obtenido se aprecia que la intervención se realiza correctamente (caída de la línea negra, la cual indica el tamaño total de la población) pero tras ella se produce un crecimiento extremadamente rápido. Esto se debe a que la tasa de muerte de las células se hace muy pequeña, por ello en los lugares del código donde se calculaba la tasa de muerte se le introdujo un pequeño cambio de tener : death_rate = log(1 + N/K) se pasó a a tener death_rate = max(1, log(1 + N/K)) y de esta forma se aseguraba que la tasa de muerte no bajara de 1 en ningún momento, produciendo resultados más acordes con los esperados.

4

Pruebas Realizadas y Resultados

4.1. Pruebas realizadas

4.1.1. El paquete Testthat

Testthat es un paquete de R creado por Hadley Wickham [16] que facilita mucho la creación y utilización de tests, ya que simplifica la forma en la que se codifican los mismos.
En el siguiente ejemplo se puede apreciar la facilidad de su uso:

Código 4.1: Ejemplo de uso Testthat.

```
1 test_that("str_length_es_el_numero_de_caracteres", {
2   expect_equal(str_length("a"), 1)
3   expect_equal(str_length("ab"), 2)
4   expect_equal(str_length("abc"), 3)
5 })
```

test_that("nombre_test",) se utiliza para crear el test, las funciones ***expect_equal*** para verificar si el resultado es igual al esperado (existen diversas funciones expect: equal, match, identical, true, warning... y varias más), en este caso se comprueba que la longitud de una cadena es igual al número de caracteres para diferentes tamaños de cadena.

4.1.2. Tests ya implementados

Antes del comienzo de este proyecto OncoSimulR ya contaba con un gran número de tests, más de **2000** tests automáticos que prueban casi por completo el código de OncoSimulR, estos tests realizan una cobertura de código de aproximadamente el 94 %. Por otro lado existen otros tests que se ejecutan de forma manual ya que son tests cuya duración puede llevar horas y realizan comprobaciones más exhaustivas probando casos poco comunes (eventos de baja probabilidad). Estos test aumentan un poco más la cobertura de código.

Una vez implementadas las intervenciones dentro del código era necesario que el paquete con los nuevos cambios superase todos los tests. Para ello, tras ejecutarlos por primera vez, se detectaron los errores producidos por el código nuevo. Por ejemplo, dos tests que esperaban un resultado determinado ya no funcionaban porque al modificar el código del modelo de McFarland el resultado de la simulación cambiaba y por tanto fue necesaria la actualización de los valores esperados por estos tests. Esto se hizo así ya que casi todos lo errores en los tests se debieron al cambio realizado en el death_ rate(), comentado al final de Apartado 3.3, dado que este valor ya no baja de 1 y antes si existía esa posibilidad y por tanto era necesario cambiar el test. Una vez detectados los errores se hicieron las modificaciones necesarias (tanto en el código como en los propios test, dependiendo del error) para solucionar todos los problemas de tal forma que los tests siguieran realizando su función y que se superara el 100 % de los tests. En la Figura 4.1 se puede ver la salida tras la ejecución de todos los test:

```
══ testthat results  ═══════════════
OK: 3224 SKIPPED: 12 FAILED: 0
✔ |  0       | OncoSimulR [293.3 s]
```

Figura 4.1: Salida obtenida tras la ejecución de todos los tests.

Por tanto al haber superado todos los tests implementados antes de la realización de las intervenciones, implica que se ha realizado una prueba de regresión permitiendo demostrar que el código sigue funcionando correctamente a pesar de los cambios introducidos.

4.1.3. Tests propios

Dado que se ha añadido funcionalidad nueva es necesario comprobar que esta funciona correctamente y produce los cambios deseados. Por ello se han implementado diferentes tests que comprueban que el tamaño de la población tras la ejecución de una o varias intervenciones tiene el valor esperado. Se han realizado tests con un número diferente de intervenciones y con diferente orden para cada uno de los modelos de crecimiento. Las combinaciones serían:

- Tests con 1 intervención.
  - Modelo exponencial.
  - Modelo de McFarland.
- Tests con 3 intervenciones
  - Modelo exponencial.
    - Repetidas.
    - Ordenadas.

- ◇ Desordenadas.

- Modelo de McFarland.
  - ◇ Repetidas.
  - ◇ Ordenadas.
  - ◇ Desordenadas.

En todos los casos se verifica que la intervención se produce el momento correcto, cuando se alcanza el tamaño de población determinado en el "trigger", y que la población tras la intervención es la que debe tras eliminar el porcentaje de células indicado en la acción. En el siguiente código se muestra la implementación de uno de los test creados para comprobar el correcto funcionamiento de las intervenciones.

Código 4.2: Ejemplo de test con 1 intervención para el modelo exponencial.

```
test_that("Population_after_one_intervention_Exp",{
  popSize <- 5000
  fractionPopSize <- 0.15
  sampleEvery <- 0.01
  modelChanges <- list( Change1 = list( trigger = list(popSize = popSize),
                                        action = list(fractionPopSize = fractionPopSize)))
  nd <- 70
  np <- 0
  s <- 0.1
  sp <- 1e-3
  spp <- -sp/(1 + sp)
  mcf1 <- allFitnessEffects(noIntGenes = c(rep(s, nd), rep(spp, np)), drvNames = seq.int(nd))
  s1 <- oncoSimulIndiv(mcf1,
                       model = "Exp",
                       mu = 1e-5,
                       detectionProb = NA,
                       detectionSize = 1e4,
                       detectionDrivers = NA,
                       sampleEvery = sampleEvery,
                       keepEvery = 0.01,
                       initSize = 1000,
                       finalTime = 2000,
                       onlyCancer = FALSE,
                       modelChanges = modelChanges)
  p_time <- s1$pops.by.time
  df <- as.data.frame(p_time) %>% filter(V1 %in% (s1$other$interv_time))
  pop_after_interv <- rowSums(df[,2:ncol(df)])
  fraction_pop <- s1$other$interv_pop *fractionPopSize
  expect_true(pop_after_interv <= fraction_pop)
})
```

El resto de tests implementados se encuentran en Apéndice A.

4.2. Resultados

4.2.1. Simulaciones realizadas con oncoSimulIndiv

Los resultados finales se han obtenido mediante la realización de simulaciones usando la función oncoSimulIndiv sobre diferentes casos de ejemplo. En este caso se ha simulado: sin aplicar una intervención, aplicando una intervención del 85 %, aplicando una intervención del 90 % y aplicando una intervención del 99 %. Cada uno de los casos anteriores se ha simulado con los dos modelos de crecimiento: exponencial y de McFarland.

En la Figura 4.2 se muestra el crecimiento de la población de células a través del tiempo siguiendo el modelo exponencial. Dado que no se le aplica ninguna intervención la población crece hasta que se alcanza el tamaño de población por el cual se detecta cáncer (10e6). En las figuras: Figura 4.3, Figura 4.4, Figura 4.5 se aplica una intervención del 85 %, 90 % y 99 % respectivamente. Se puede apreciar a simple vista el momento en el que se produce y las consecuencias en el tamaño de la población. Tras esto la población vuelve a crecer exponencialmente detectando cáncer en los tres casos. También se puede observar la similitud con la Figura 1.1 en la que se muestran dos intervenciones quirúrgicas.

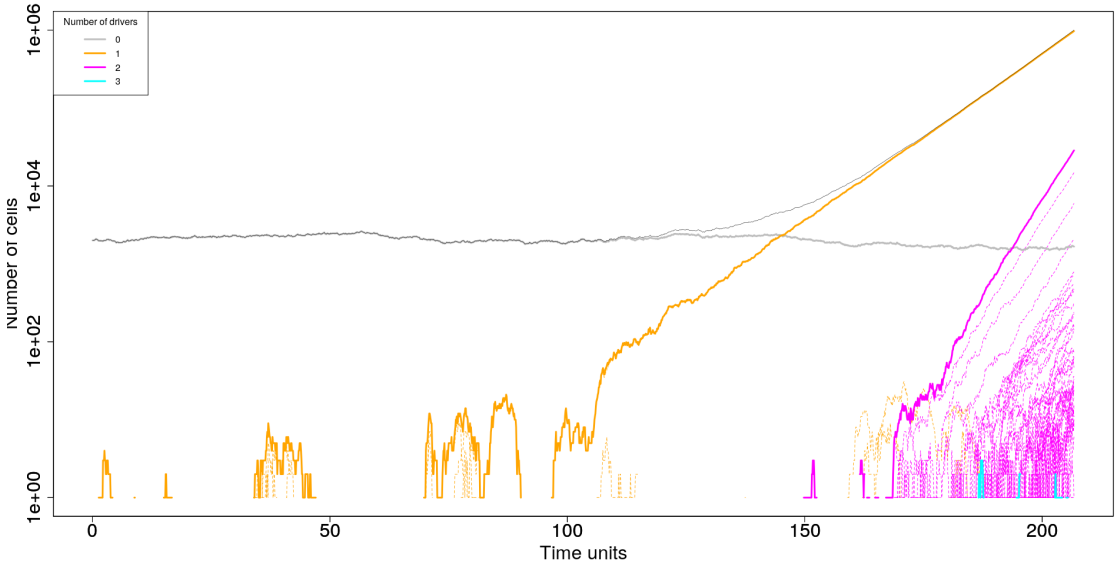


Figura 4.2: Simulación del modelo Exponencial sin intervención

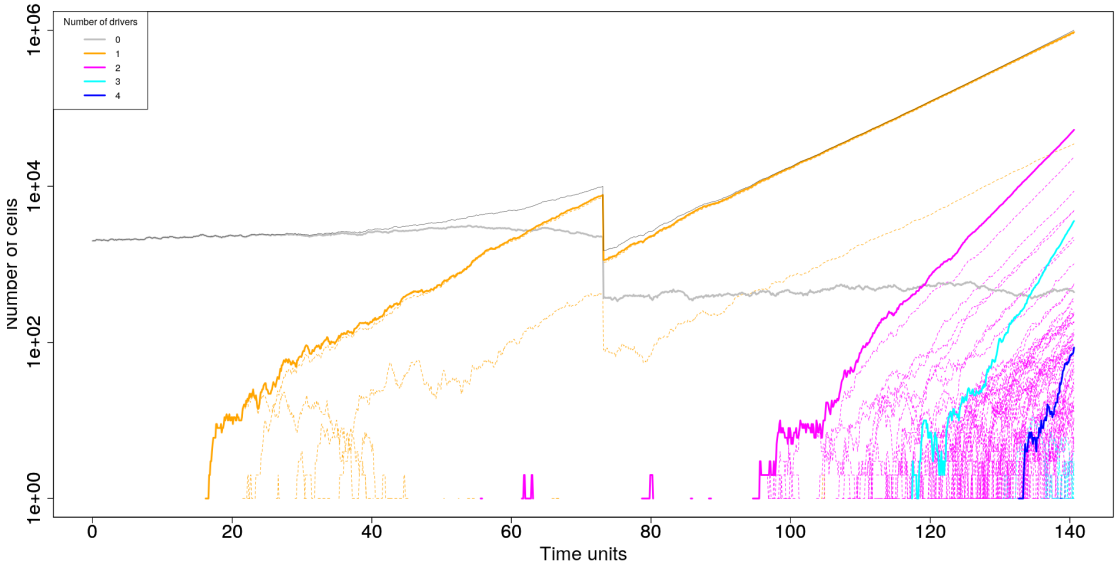


Figura 4.3: Simulación del modelo Exponencial con una intervención del 85 % al llegar a 5 veces la población inicial.

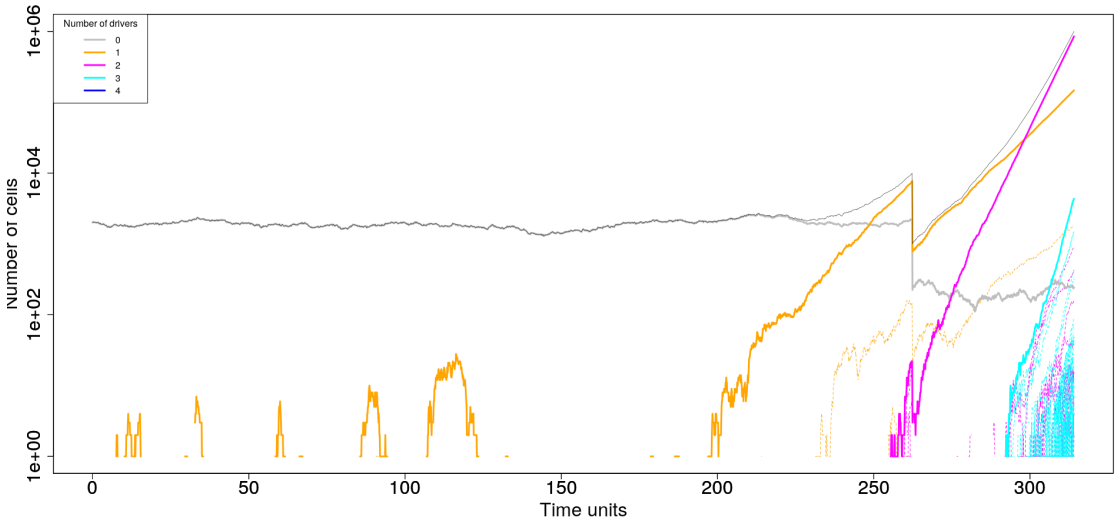


Figura 4.4: Simulación del modelo Exponencial con una intervención del 90 % al llegar a 5 veces la población inicial.

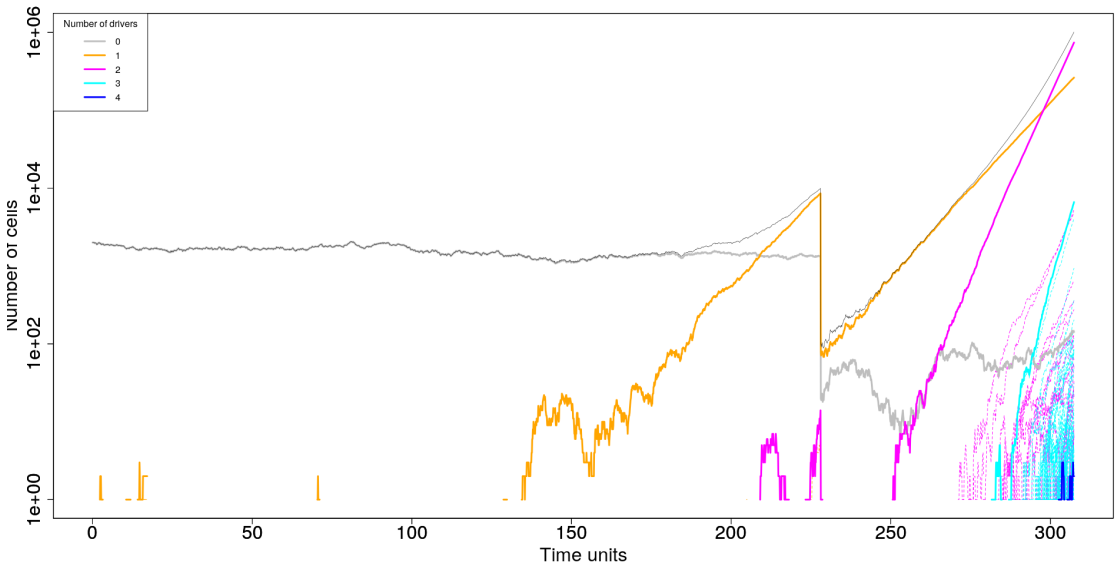


Figura 4.5: Simulación del modelo Exponencial con una intervención del 99 % al llegar a 5 veces la población inicial.

En cuanto a las simulaciones realizadas con McFarland en la Figura 4.6 se simula sin aplicar una intervención. Se puede apreciar como el crecimiento es más lento que en el modelo exponencial ya que se mantiene equilibradas la tasa de nacimiento y la tasa de muertes. Cuando aplicamos una intervención como se muestra en la Figura 4.7, Figura 4.8, Figura 4.9 se producen los mismos efectos que en las gráficas del modelo exponencial pero es más complicado de apreciar debido al gran número de lineas de las gráficas, por ello se marca con la flecha roja la posición de la intervención. La recuperación es más rápida que en el modelo exponencial porque en este caso en ese momento de la simulación ya se tienen células con capacidad de crecer muy deprisa y por eso el tumor se recupera tan rápido.

4.2.2. Estadísticas a partir de 200 simulaciones

En este apartado se analizarán algunas de las estadísticas que se pueden obtener al usar la función oncoSimulPop, detallada en el Apartado 2.4, que realiza “N” simulaciones.
En todos los casos las tablas representan la media calculada de cada estadística tras 200 simulaciones. Cada conjunto de 200 simulaciones se ha ejecutado con cada modelo de crecimiento (exponencial y McFarland) y con tres cortes como en casos anteriores (85 %, 90 %, 99 %). El “trigger” se activa una vez el tamaño de la población alcanza 5 veces su valor inicial (2000). En la Tabla 4.1 se muestra que porcentaje del total de 200 simulaciones han finalizado detectando cáncer. Cabe destacar la alta proporción que se obtiene en todos los casos.

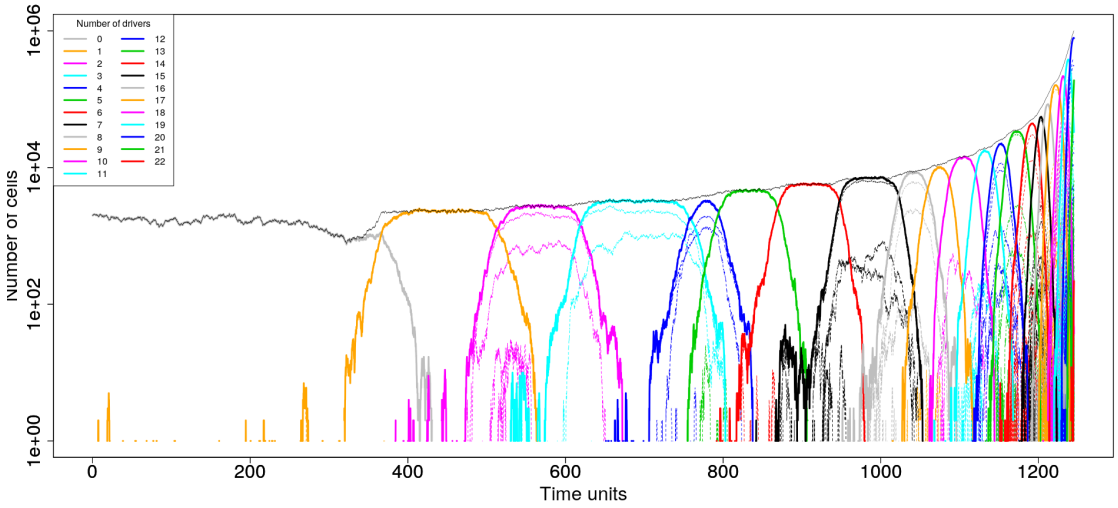


Figura 4.6: Simulación del modelo de McFarland sin intervención.

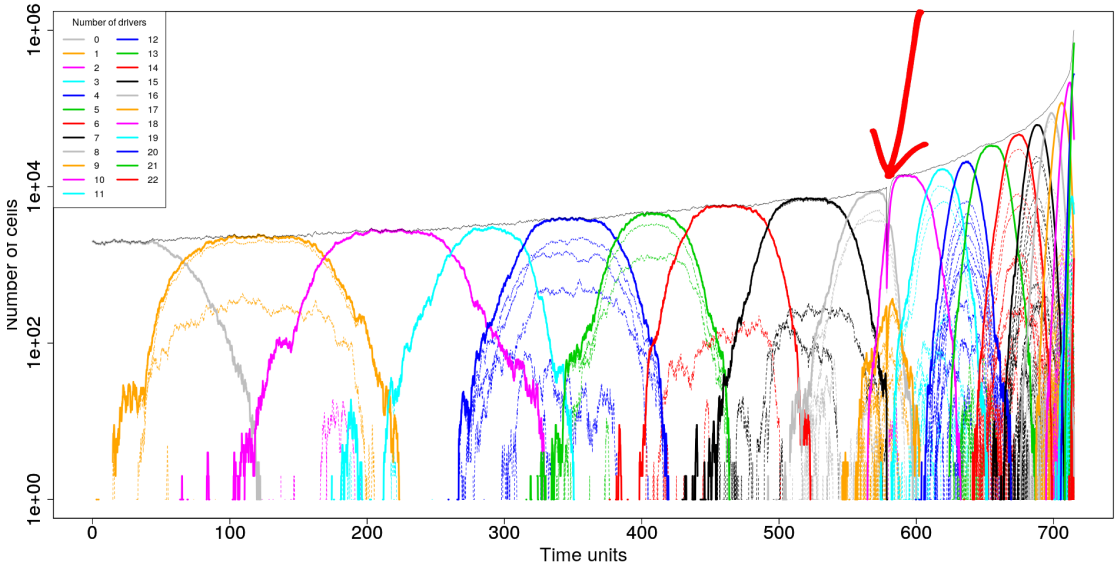


Figura 4.7: Simulación del modelo de McFarland con una intervención del 85 % al llegar a 5 veces la población inicial.

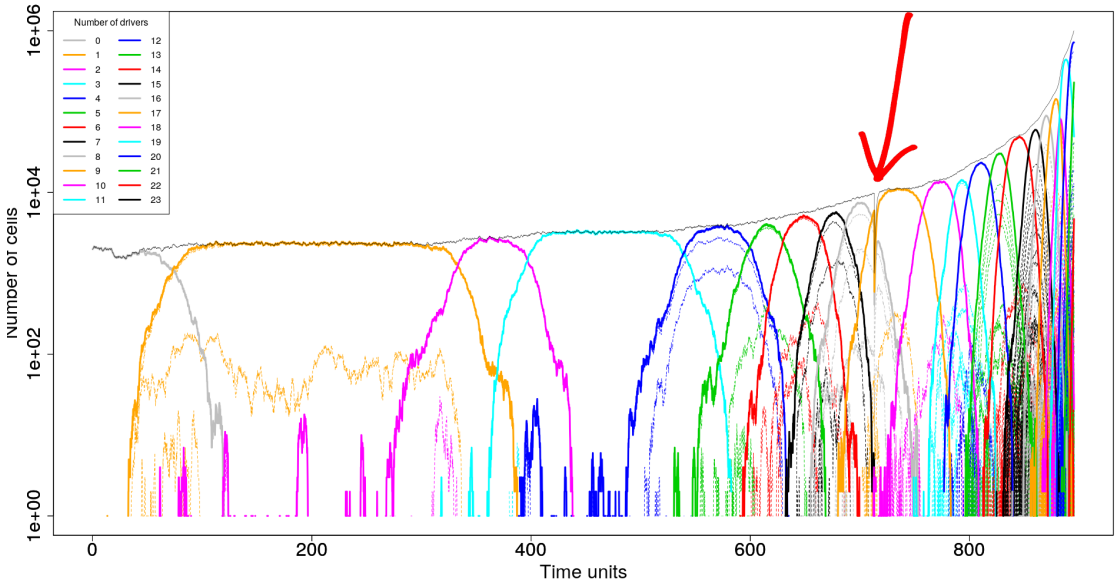


Figura 4.8: Simulación del modelo de McFarland con una intervención del 90 % al llegar a 5 veces la población inicial.

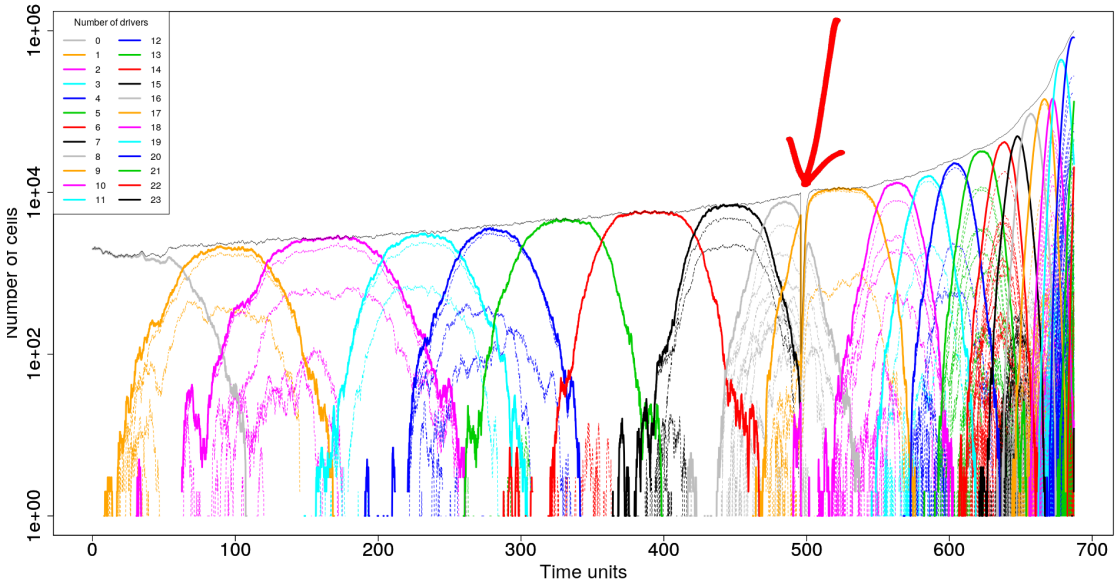


Figura 4.9: Simulación del modelo de McFarland con una intervención del 99 % al llegar a 5 veces la población inicial.

Modelo	% eliminado en intervención	% que acaban en cáncer
Modelo exponencial	NO	0.975
	85	0.990
	90	0.985
	99	0.995
Modelo de McFarland	NO	0.965
	85	0.970
	90	0.990
	99	0.990

Tabla 4.1: Proporción de las simulaciones que acaban en cáncer.

En la Tabla 4.2 y la Tabla 4.3 se puede apreciar que según aumentamos el porcentaje de población eliminado durante la intervención los tiempos aumentan en todos los casos. En los casos concretos pertenecientes al modelo exponencial este diferencia es más sustancial que la diferencia que se aprecia en el modelo de McFarland.

Modelo	% eliminado en intervención	Tiempo para cáncer
Modelo exponencial	NO	239.016
	85	236.870
	90	257.362
	99	275.232
Modelo de McFarland	NO	1006.023
	85	1005.721
	90	1009.958
	99	1038.497

Tabla 4.2: Tiempo necesario para llegar a cáncer.

Modelo	% eliminado en intervención	Tiempo recuperación tras intervención
Modelo exponencial	NO	0.000
	85	20.232
	90	23.914
	99	47.545
Modelo de McFarland	NO	0.000
	85	9.720
	90	9.921
	99	9.921

Tabla 4.3: .Tiempo para recuperar tamaño de población previo a la intervención.

Estos resultados son los esperados ya que cuanto mayor sea la cantidad de población eliminada, más tiempo se tardará en recuperar la proporción de células y más tiempo será necesario para que acaben en cáncer. Por tanto se puede decir que las simulaciones en las que se aplica una intervención frenan momentáneamente el crecimiento del tumor comparando con una simulación en la que no se aplica nada.

5

Conclusiones y trabajo futuro

En esta sección se detallan las conclusiones obtenidas y se analiza si se han alcanzado los objetivos que se marcaron al inicio de este proyecto así como las opciones de desarrollo futuras en las que se podría trabajar.

5.1. Conclusiones

El objetivo principal de este proyecto era el de implementar mediante R y C++ una funcionalidad que simulase la realización de una intervención quirúrgica en las simulaciones de OncoSimulR. Permitiendo al usuario especificar cuándo se produce la intervención y las consecuencias de la misma.

Se puede afirmar que este y el resto de los objetivos propuestos en el Apartado 1.2 se han cumplido ya que a la vista de los resultados expuestos a lo largo de esta memoria se pueden observar los diferentes hitos alcanzados durante todo el desarrollo.

Existen diversas posibilidades para la continuación y ampliación de este proyecto que se describen en más detalle en el Apartado 5.2.

Por último comentar que el repositorio con todo el código realizado esta disponible en `https://github.com/DiegoChG/OncoSimul`, es público y cualquiera puede descargar, instalar y probar el código implementado. En el futuro se integrará con el código de OncoSimulR de Ramón Diaz Uriarte `https://github.com/rdiaz02/OncoSimul` ampliando así las funciones del paquete y permitiendo que los usuarios interesados puedan realizar simulaciones aplicando intervenciones y sirve como base para la implementación de las posibles ampliaciones futuras. Por otro lado también se ha modificado la documentación, actualizando la información de las funciones modificadas durante la codificación de las intervenciones, siguiendo los estándares de los paquetes R/BioCondutor.

5.2. Trabajo futuro

A partir del trabajo realizado en este proyecto, el posible trabajo futuro sería la ampliación de los tipos de intervenciones, es decir, se podrían añadir nuevos triggers y acciones. Esto daría mas versatilidad y se ampliarían los posibles casos a simular con las intervenciones.
Algunos ejemplos de nuevos triggers y acciones serían los siguientes:

- **Trigger de tiempo:** La intervención se activará cuando se alcance un determinado valor de tiempo de ejecución.
- **Trigger de genotipo:** Si un genotipo alcanza un determinado tamaño dentro de la población la intervención se activa. Similar a la intervención quirúrgica pero condicionada a la presencia de determinados clones.
- **Acción de cambio de fitness:** Modificación del fitness de un determinado genotipo mediante la modificación del fitness landscape cuando la población alcanza un tamaño especificado. Este caso sería similar a simular la aplicación de quimioterapia.

También habría que tener en cuenta la forma en la que se comprueban la activación o no de las intervenciones, ya que se deberían activar si se cumplen todos los triggers y en ese caso realizar todas las acciones que correspondan a esa intervención.

```
modelChanges = list(
        Change1 = list(
                trigger = list(popsize = 1000, # trigger de población
                                genotype["A, E"] = 100, # trigger de genotipo
                                time = 10), # trigger de tiempo
                action = list(fractionPopSize = 0.2, # acción de cambio de tamaño
                                fitnessLandscape = fl1) # acción de cambio de fitness
                ),
        Change2 = list(
                trigger = list(popsize = 10000, # trigger de población
                                genotype['A, B"] = 1000, # trigger de genotipo
                                time = 30), # trigger de tiempo
                action = list(fractionPopSize = 0.9, # acción de cambio de tamaño
                                fitnessLandscape = fl2) # acción de cambio de fitness
))
```

Cuadro 5.1: Ejemplo de intervención ampliada, con los nuevos tipos de triggers y acciones (indicados en los comentarios)

Bibliografía

[1] ASCO - American Society of Clinical Oncology, "What is Cancer? | Cancer.Net." https://www.cancer.net/navigating-cancer-care/cancer-basics/what-cancer.

[2] N. fundation, "Paul Ehrlich - Biographical." https://www.nobelprize.org/nobel_prizes/medicine/laureates/1908/ehrlich-bio.html.

[3] ASCO, "How Cancer is Treated | Cancer.Net." https://www.cancer.net/navigating-cancer-care/how-cancer-treated.

[4] H. Sbeity, "Review of Optimization Methods for Cancer Chemotherapy Treatment Planning," *Journal of Computer Science & Systems Biology*, vol. 8, no. 2, 2015. https://bit.ly/2K3uI73.

[5] J. G. Reiter, I. Bozic, K. Chatterjee, and M. A. Nowak, "TTP: Tool for Tumor Progression," pp. 101–106, Springer, Berlin, Heidelberg, 2013. http://pub.ist.ac.at/ttp/resources/reiter_ttp_tr2013.pdf.

[6] F. Zanini and R. A. Neher, "FFPopSim: an efficient forward simulation package for the evolution of large populations," *Bioinformatics*, vol. 28, pp. 3332–3333, dec 2012. https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/bts633.

[7] B. Peng and M. Kimmel, "simuPOP: a forward-time population genetics simulation environment," *Bioinformatics*, vol. 21, pp. 3686–3687, sep 2005. https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/bti584.

[8] R. Diaz-Uriarte, "OncoSimulR: Genetic simulation with arbitrary epistasis and mutator genes in asexual populations," *Bioinformatics*, 2017. https://academic.oup.com/bioinformatics/article/33/12/1898/2982052.

[9] R. Diaz-uriarte, "OncoSimulR : forward genetic simulation in asexual populations with arbitrary epistatic interactions and a focus on modeling tumor progression .," 2015. https://rdiaz02.github.io/OncoSimul/OncoSimulR.html.

[10] R. Diaz-Uriarte, "Identifying restrictions in the order of accumulation of mutations during tumor progression: effects of passengers, evolutionary models, and sampling," *BMC Bioinformatics*, vol. 16, p. 41, dec 2015. http://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-015-0466-7.

[11] R. Diaz-Uriarte, "Cancer progression models and fitness landscapes: a many-to-many relationship," *Bioinformatics*, vol. 34, no. 5, pp. 836–844, 2018. https://academic.oup.com/bioinformatics/article/34/5/836/4557185.

[12] W. H. Mather, J. Hasty, and L. S. Tsimring, "Fast stochastic algorithm for simulating evolutionary population dynamics," *Bioinformatics*, vol. 28, pp. 1230–1238, may

2012. https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/bts130.

[13] Rocker, "rocker/rstudio - Docker Hub." https://hub.docker.com/u/rocker/.

[14] Docker, "What is Docker?." https://www.docker.com/what-docker.

[15] D. Eddelbuettel and C. Sanderson, "Rcpparmadillo: Accelerating r with high-performance c++ linear algebra," *Computational Statistics and Data Analysis*, vol. 71, pp. 1054–1063, March 2014. https://github.com/RcppCore/RcppArmadillo/blob/master/inst/include/RcppArmadilloExtensions/rmultinom.h.

[16] Hadley Wickham, "Testing · R packages." http://r-pkgs.had.co.nz/tests.html.

[17] Docker, "Dockerfile | Docker Documentation," 2018. https://docs.docker.com/engine/reference/builder/.

Definiciones

fitness Eficacia biológica. Probabilidad de éxito al reproducirse de un genotipo.

fitness landscapes Mapeo en el que se asocia a cada genotipo un fitness.

genomas El genoma es el conjunto de instrucciones genéticas que se encuentra en una célula.

genotipo Un genotipo es la colección de genes de un individuo, el término también puede referirse a los dos alelos heredados de un gen en particular.

modelo de McFarland Modelo de crecimiento en el que la tasa de muerte aumenta con el número de células; da lugar a un modelo que es parecido al crecimiento logísitico, con una capacidad de carga (carrying capacity) que puede aumentar si aparecen células con una tasa de nacimiento mayor.

modelo exponencial Modelo de crecimiento en el que el aumento de la población se produce de manera exponencial.

Anexos

A

Implementación de los tests

Código de los tests creados para probar el correcto funcionamiento de las intervenciones.

- Test: Población correcta tras una intervención utilizando el modelo de McFarland.

```
test_that("Population_after_one_intervention_McFL",{
  popSize <- 5000
  fractionPopSize <- 0.15
  sampleEvery <- 0.01
  modelChanges <- list( Change1 = list( trigger = list(popSize = popSize),
                                        action = list(fractionPopSize = fractionPopSize)))
  nd <- 70
  np <- 0
  s <- 0.1
  sp <- 1e-3
  spp <- -sp/(1 + sp)
  mcf1 <- allFitnessEffects(noIntGenes = c(rep(s, nd), rep(spp, np)),
                            drvNames = seq.int(nd))
  s1 <- oncoSimulIndiv(mcf1,
                       model = "McFL",
                       mu = 1e-5,
                       detectionProb = NA,
                       detectionSize = 1e4,
                       detectionDrivers = NA,
                       sampleEvery = sampleEvery,
                       keepEvery = 0.01,
                       initSize = 1000,
                       finalTime = 2000,
                       onlyCancer = FALSE,
                       modelChanges = modelChanges)
  p_time <- s1$pops.by.time
  # Take the pop size after intervention
  df <- as.data.frame(p_time) %>%
    filter(V1 %in% (s1$other$interv_time))
  pop_after_interv <- rowSums(df[,2:ncol(df)])
  # Obtain the expected value of pop
  fraction_pop <- s1$other$interv_pop *fractionPopSize
  expect_true(pop_after_interv <= fraction_pop)
})
```

- Test: Población correcta tras 3 intervenciones repetidas utilizando el modelo exponencial.

```
test_that("Population after 3 repeated interventions Exp",{
  popSize <- 6000
  fractionPopSize <- 0.10
  sampleEvery <- 0.01
  modelChanges = list( Change1 = list( trigger = list(popSize = popSize),
                                       action = list(fractionPopSize = fractionPopSize)),
                       Change2 = list( trigger = list(popSize = popSize),
                                       action = list(fractionPopSize = fractionPopSize)),
                       Change3 = list( trigger = list(popSize = popSize),
                                       action = list(fractionPopSize = fractionPopSize)))
  nd <- 70
  np <- 0
  s <- 0.1
  sp <- 1e-3
  spp <- -sp/(1 + sp)
  mcf1 <- allFitnessEffects(noIntGenes = c(rep(s, nd), rep(spp, np)), drvNames = seq.int(nd))
  s1 <- oncoSimulIndiv(mcf1,
                       model = "Exp",
                       mu = 1e-5,
                       detectionProb = NA,
                       detectionSize = 1e4,
                       detectionDrivers = NA,
                       sampleEvery = sampleEvery,
                       keepEvery = 0.001,
                       initSize = 1000,
                       finalTime = 2000,
                       onlyCancer = FALSE,
                       modelChanges = modelChanges)
  p_time <- s1$pops.by.time
  # Take the pop size after intervention
  df <- as.data.frame(p_time) %>%
    filter(V1 %in% (s1$other$interv_time))
  pop_after_interv <- rowSums(df[,2:ncol(df)])
  # Obtain the expected value of pop
  fraction_pop <- s1$other$interv_pop *fractionPopSize
  expect_true(all(pop_after_interv <= fraction_pop))
})
```

- Test: Población correcta tras 3 intervenciones ordenadas utilizando el modelo exponencial.

```
test_that("Population after 3 ordered interventions Exp",{
  popSize1 <- 4000
  fractionPopSize1 <- 0.15
  popSize2 <- 7000
  fractionPopSize2 <- 0.1
  popSize3 <- 10000
  fractionPopSize3 <- 0.01
  sampleEvery <- 0.01
  modelChanges = list( Change1 = list( trigger = list(popSize = popSize1),
                                       action = list(fractionPopSize = fractionPopSize1)),
                       Change2 = list( trigger = list(popSize = popSize2),
                                       action = list(fractionPopSize = fractionPopSize2)),
                       Change3 = list( trigger = list(popSize = popSize3),
                                       action = list(fractionPopSize = fractionPopSize3)))
  nd <- 70
  np <- 0
  s <- 0.1
  sp <- 1e-3
  spp <- -sp/(1 + sp)
  mcf1 <- allFitnessEffects(noIntGenes = c(rep(s, nd), rep(spp, np)), drvNames = seq.int(nd))
  s1 <- oncoSimulIndiv(mcf1,
                       model = "Exp",
                       mu = 1e-5,
                       detectionProb = NA,
                       detectionSize = 1.5e4,
                       detectionDrivers = NA,
                       sampleEvery = sampleEvery,
                       keepEvery = 0.01,
                       initSize = 1000,
                       finalTime = 2000,
                       onlyCancer = FALSE,
                       modelChanges = modelChanges)
  p_time <- s1$pops.by.time
  df <- as.data.frame(p_time) %>% filter(V1 %in% (s1$other$interv_time)) # Take the pop size
      after intervention
  pop_after_interv <- rowSums(df[,2:ncol(df)])
  # Obtain the expected value of pop
  fraction_pop <- c(s1$other$interv_pop[1] *fractionPopSize1, s1$other$interv_pop[2] *fractionPopSize2,
                    s1$other$interv_pop[3] *fractionPopSize3)
  expect_true(all(pop_after_interv <= fraction_pop))
})
```

- Test: Población correcta tras 3 intervenciones desordenadas utilizando el modelo exponencial.

```
test_that("Population after 3 disordered interventions Exp",{
  popSize1 <- 4000
  fractionPopSize1 <- 0.15
  popSize2 <- 6000
  fractionPopSize2 <- 0.1
  popSize3 <- 2700
  fractionPopSize3 <- 0.15
  sampleEvery <- 0.01
  modelChanges = list( Change1 = list( trigger = list(popSize = popSize1),
                                       action = list(fractionPopSize = fractionPopSize1)),
                       Change2 = list( trigger = list(popSize = popSize2),
                                       action = list(fractionPopSize = fractionPopSize2)),
                       Change3 = list( trigger = list(popSize = popSize3),
                                       action = list(fractionPopSize = fractionPopSize3)))
  nd <- 70
  np <- 0
  s <- 0.1
  sp <- 1e-3
  spp <- -sp/(1 + sp)
  mcf1 <- allFitnessEffects(noIntGenes = c(rep(s, nd), rep(spp, np)), drvNames = seq.int(nd))
  s1 <- oncoSimulIndiv(mcf1,
                       model = "Exp",
                       mu = 1e-5,
                       detectionProb = NA,
                       detectionSize = 1e4,
                       detectionDrivers = NA,
                       sampleEvery = sampleEvery,
                       keepEvery = 0.01,
                       initSize = 2000,
                       finalTime = 2000,
                       onlyCancer = FALSE,
                       modelChanges = modelChanges)
  p_time <- s1$pops.by.time
  df <- as.data.frame(p_time) %>% filter(V1 %in% (s1$other$interv_time))# Take the pop size
      after intervention
  pop_after_interv <- rowSums(df[,2:ncol(df)])
  # Obtain the expected value of pop
  fraction_pop <- c(s1$other$interv_pop[1] *fractionPopSize3, s1$other$interv_pop[2] *fractionPopSize1,
                    s1$other$interv_pop[3] *fractionPopSize2)
  expect_true(all(pop_after_interv <= fraction_pop))
})
```

- Test: Población correcta tras 3 intervenciones repetidas utilizando el modelo de McFarland.

```
test_that("Population after 3 repeated interventions McFL",{
  popSize <- 6000
  fractionPopSize <- 0.10
  sampleEvery <- 0.01
  modelChanges = list( Change1 = list( trigger = list(popSize = popSize),
                                       action = list(fractionPopSize = fractionPopSize)),
                       Change2 = list( trigger = list(popSize = popSize),
                                       action = list(fractionPopSize = fractionPopSize)),
                       Change3 = list( trigger = list(popSize = popSize),
                                       action = list(fractionPopSize = fractionPopSize)))
  nd <- 70
  np <- 0
  s <- 0.1
  sp <- 1e-3
  spp <- -sp/(1 + sp)
  mcf1 <- allFitnessEffects(noIntGenes = c(rep(s, nd), rep(spp, np)),
                            drvNames = seq.int(nd))
  s1 <- oncoSimulIndiv(mcf1,
                       model = "Exp",
                       mu = 1e-5,
                       detectionProb = NA,
                       detectionSize = 1e4,
                       detectionDrivers = NA,
                       sampleEvery = sampleEvery,
                       keepEvery = 0.01,
                       initSize = 1000,
                       finalTime = 2000,
                       onlyCancer = FALSE,
                       modelChanges = modelChanges)
  p_time <- s1$pops.by.time

  df <- as.data.frame(p_time) %>% filter(V1 %in% (s1$other$interv_time)) # Take the pop size
      after intervention
  pop_after_interv <- rowSums(df[,2:ncol(df)])
  # Obtain the expected value of pop
  fraction_pop <- s1$other$interv_pop *fractionPopSize
  expect_true(all(pop_after_interv <= fraction_pop))
})
```

- Test: Población correcta tras 3 intervenciones ordenadas utilizando el modelo de McFarland.

```
test_that("Population after 3 ordered interventions McFL",{
  popSize1 <- 2700
  fractionPopSize1 <- 0.15
  popSize2 <- 4000
  fractionPopSize2 <- 0.1
  popSize3 <- 5000
  fractionPopSize3 <- 0.15
  sampleEvery = 0.01
  modelChanges = list( Change1 = list( trigger = list(popSize = popSize1),
                                       action = list(fractionPopSize = fractionPopSize1)),
                       Change2 = list( trigger = list(popSize = popSize2),
                                       action = list(fractionPopSize = fractionPopSize2)),
                       Change3 = list( trigger = list(popSize = popSize3),
                                       action = list(fractionPopSize = fractionPopSize3)))
  nd <- 70
  np <- 0
  s <- 0.1
  sp <- 1e-3
  spp <- -sp/(1 + sp)
  mcf1 <- allFitnessEffects(noIntGenes = c(rep(s, nd), rep(spp, np)), drvNames = seq.int(nd))
  s1 <- oncoSimulIndiv(mcf1,
                       model = "Exp",
                       mu = 1e-5,
                       detectionProb = NA,
                       detectionSize = 1e4,
                       detectionDrivers = NA,
                       sampleEvery = sampleEvery,
                       keepEvery = 0.01,
                       initSize = 2000,
                       finalTime = 2000,
                       onlyCancer = FALSE,
                       modelChanges = modelChanges)
  p_time <- s1$pops.by.time

  df <- as.data.frame(p_time) %>% filter(V1 %in% (s1$other$interv_time))# Take the pop size
      after intervention
  pop_after_interv <- rowSums(df[,2:ncol(df)])
  # Obtain the expected value of pop
  fraction_pop <- c(s1$other$interv_pop[1] *fractionPopSize1, s1$other$interv_pop[2] *fractionPopSize2,
                    s1$other$interv_pop[3] *fractionPopSize3)
  expect_true(all(pop_after_interv <= fraction_pop))
})
```

- Test: Población correcta tras 3 intervenciones desordenadas utilizando el modelo de McFarland.

```
test_that("Population after 3 disordered interventions McFL",{
  popSize1 <- 4000
  fractionPopSize1 <- 0.15
  popSize2 <- 6000
  fractionPopSize2 <- 0.1
  popSize3 <- 2700
  fractionPopSize3 <- 0.15
  sampleEvery = 0.01
  modelChanges = list( Change1 = list( trigger = list(popSize = popSize1),
                                       action = list(fractionPopSize = fractionPopSize1)),
                       Change2 = list( trigger = list(popSize = popSize2),
                                       action = list(fractionPopSize = fractionPopSize2)),
                       Change3 = list( trigger = list(popSize = popSize3),
                                       action = list(fractionPopSize = fractionPopSize3)))
  nd <- 70
  np <- 0
  s <- 0.1
  sp <- 1e-3
  spp <- -sp/(1 + sp)
  mcf1 <- allFitnessEffects(noIntGenes = c(rep(s, nd), rep(spp, np)), drvNames = seq.int(nd))
  s1 <- oncoSimulIndiv(mcf1,
                       model = "Exp",
                       mu = 1e-5,
                       detectionProb = NA,
                       detectionSize = 1e4,
                       detectionDrivers = NA,
                       sampleEvery = sampleEvery,
                       keepEvery = 0.01,
                       initSize = 1000,
                       finalTime = 2000,
                       onlyCancer = FALSE,
                       modelChanges = modelChanges)
  p_time <- s1$pops.by.time
  df <- as.data.frame(p_time) %>% filter(V1 %in% (s1$other$interv_time)) # Take the pop size
      after intervention
  pop_after_interv <- rowSums(df[,2:ncol(df)])
  # Obtain the expected value of pop
  fraction_pop <- c(s1$other$interv_pop[1] *fractionPopSize3, s1$other$interv_pop[2] *fractionPopSize1,
                    s1$other$interv_pop[3] *fractionPopSize2)
  expect_true(all(pop_after_interv <= fraction_pop))
})
```

B

Instalación de OncoSimulR mediante Dockerfile

En el caso de este proyecto se ha utilizado un Dockerfile [17] para automatizar la instalación de OncosimulR en un contenedor de Docker.

1.– Se ejecuta ***docker build -t oncosimul*** en la carpeta donde se encuentren los siguientes ficheros:

```
FROM rocker/rstudio:latest
RUN apt-get update
RUN apt-get install libxml2-dev -y
ADD inicial.R /var/inicial/
RUN Rscript /var/inicial/inicial.R
CMD [/init"]
```

Cuadro B.1: Dockerfile

```
source("https://bioconductor.org/biocLite.R")
biocLite("OncoSimulR")
```

Cuadro B.2: Inicial.R

De esta forma se crea una imagen con todo lo necesario para crear un contenedor en el cual tendremos un entorno de desarrollo basado en RStudio con el paquete OncoSimulR ya instalado.

2.– Se ejecuta ***docker run -d -e PASSWORD= «password» -p 8787:8787 -it –name oncosimul -v /Documents/Docker/shared:/home/rstudio/shared oncosimul*** y así se genera un contenedor a partir de la imagen creada anteriormente que tendrá una carpeta compartida con el host (localizada en este caso en /Documents/Docker/shared).

3.– Para iniciar el contenedor se ejecuta: ***docker start «nombre_contenedor»*** y para acceder a él y comenzar a utilizar RStudio se utiliza la siguiente URL en el navegador:

http://localhost:8787

4.– Para detener el contenedor se ejecuta: ***docker stop «nombre_contenedor»***

5.– Si queremos acceder a la terminal del contenedor ***docker exec -it «nombre_contenedor» bash***